```
//File: cameraControlTest.pde
//Author: Aaron Olson
//Written for SD1112

//This file contains the main program event loop
//for the Arduino microprocessor. Includes command
//interpretation and communication.

#include <inttypes.h>
#include <avr/pgmspace.h>

#include <avrpins.h>
#include <max3421e.h>
#include <usbhost.h>
#include <usb_ch9.h>
#include <Usb.h>
#include <usbhub.h>
#include <address.h>

#include <message.h>
#include <parsetools.h>
#include <hexdump.h>

#include <ptp.h>
#include <ptpdebug.h>

#include <simpletimer.h>
#include "pseventparser.h"
#include "ptpobjinfoparser.h"
#include "ptpobjhandleparser.h"
#include "ptpthumbparser.h"
#include "canonpschdk.h"

char incomingByte;
bool cameraConnected = false;

class CamStateHandlers : public PSStateHandlers
{
      enum CamStates { stInitial, stDisconnected, stConnected };
      CamStates stateConnected;

public:
      CamStateHandlers() : stateConnected(stInitial) {};

      virtual void OnDeviceDisconnectedState(PTP *ptp);
      virtual void OnDeviceInitializedState(PTP *ptp);
      virtual void OnDeviceBusyState(PTP *ptp);
} CamStates;

USB Usb;
USBHub Hub1(&Usb);
CanonPSCHDK PS(&Usb, &CamStates);
```

```
void CamStateHandlers::OnDeviceDisconnectedState(PTP *ptp)
{
    if (stateConnected == stConnected || stateConnected == stInitial)
    {
        stateConnected = stDisconnected;
        cameraConnected = false;

        Notify(PSTR("Camera disconnected\r\n"));
    }
}

void CamStateHandlers::OnDeviceInitializedState(PTP *ptp)
{
    if (stateConnected == stDisconnected || stateConnected == stInitial)
    {
        stateConnected = stConnected;
        cameraConnected = true;
        Notify(PSTR("Camera connected\r\n"));
    }
}

void CamStateHandlers::OnDeviceBusyState(PTP *ptp)
{
    Serial.println("I'm Busy!");
}

void setup()
{
  //Initial Code

  // Setup LED output pin
  pinMode(13, OUTPUT);

  //Setup serial read
  Serial.begin(115200);
  pinMode(2, INPUT);

  if (Usb.Init() == -1)
     Serial.println("OSC did not start.");

  //delay(2000);
}

//Main program loop
void loop()
{

  //Stalls the loop while no camera is connected.
  while(!cameraConnected)
    Usb.Task();

  //Grab byte from serial port. Only looking for one byte
  if(Serial.available() > 0)
```

```
{
  incomingByte = Serial.read();
  Serial.println(incomingByte,BYTE);
  delay(1000);
}

delay(300);

//digitalWrite(13,LOW);
//delay(2000);

//Image capture command
if(incomingByte == 'C')
{
  //digitalWrite(13,HIGH);
  Serial.println("Command received. Triggering capture!");
  incomingByte = 0;
  PS.Capture();
  //Protect it from doing anything else while the capture occurs
  delay(3000);

}
//Record mode command
else if(incomingByte == 'R')
{
  Serial.println("Command received. Switching to record mode!");
  incomingByte = 0;
  PS.SetRecordMode(1);
  delay(2000);
}
//Play mode command
else if(incomingByte == 'P')
{
  Serial.println("Command received. Switching to play mode!");
  incomingByte = 0;
  PS.SetRecordMode(0);
  delay(2000);
}
//Zoom in command
else if(incomingByte == 'Z')
{
  Serial.println("Command received. Zooming in");
  incomingByte = 0;
  PS.IncrementZoom(1);
  delay(1500);
}
//Zoom out command
else if(incomingByte == 'X')
{
  Serial.println("Command received. Zooming out");
  incomingByte = 0;
  PS.IncrementZoom(0);
  delay(1500);
```

```
  }
  //Transfer command
  else if(incomingByte == 'T')
  {
    incomingByte = 0;
    Serial.println("Command received. Parsing");
    PTPObjHandleParser prs;
    PS.GetObjectHandles(0xFFFFFFFF,0,0,&prs);
//    Serial.println("Object handles:");
//    for(int i = 0; i<16; i++)
//    {
//      Serial.println(prs.varBuffer[i],HEX);
//    }

    int bufIndex = prs.varBuffer[16] - 1;
    //Serial.println("Buffer index is:");
    //Serial.println(bufIndex,DEC);

    PS.SetRecordMode(0);
    delay(2000);

    Serial.println("Grabbing file info!");
    PTPObjInfoParser infoParse;
    PS.GetObjectInfo(prs.varBuffer[bufIndex], &infoParse);
    //Serial.println("Done grabbing file info");

    Serial.println("Thumb format:");
    Serial.println(infoParse.thumb,HEX);

    //Serial.println(0xFE,BYTE);
    //Serial.println(0xCD,BYTE);
    Serial.println("Thumb size:");
    Serial.println(infoParse.thumbSize,DEC);

    //Make sure the thumbnail is an image format
    if(0x3800<infoParse.thumb && infoParse.thumb<0x3811)
    {
      //Serial.println("Grabbing thumbnail!");
      PTPThumbParser thumbParse;
//      Serial.print(0xFE,BYTE);
//      Serial.print(0xCE,BYTE);
      Serial.println("Thumbnail transfer start");
      Serial.println(infoParse.thumbSize,DEC);
      PS.GetThumb(prs.varBuffer[bufIndex], &thumbParse);
      //May need to send Lucas an end sequence.
      Serial.println("Done grabbing thumbnail");
      //delay(5000);
    }
    else
    {
      Serial.println("Thumbnail does not exist!");
    }
```

```
    PS.SetRecordMode(1);
    delay(2000);
  }

  else if(incomingByte == 'F')
  {
    incomingByte = 0;
    Serial.println("Command received. Parsing");
    PTPObjHandleParser prs;
    PS.GetObjectHandles(0xFFFFFFFF,0,0,&prs);


    int bufIndex = prs.varBuffer[16] - 1;
    //Serial.println("Buffer index is:");
    //Serial.println(bufIndex,DEC);

    PS.SetRecordMode(0);
    delay(2000);

    Serial.println("Grabbing file info!");
    PTPObjInfoParser infoParse;
    PS.GetObjectInfo(prs.varBuffer[bufIndex], &infoParse);
    //Serial.println("Done grabbing file info");

    Serial.println("Picture format:");
    Serial.println(infoParse.pic,HEX);

    //Serial.println(0xFE,BYTE);
    //Serial.println(0xCD,BYTE);
    Serial.println("Picture size:");
    Serial.println(infoParse.picSize,DEC);

    //Make sure the thumbnail is an image format
    if(0x3800<infoParse.pic && infoParse.pic<0x3811)
    {
      //Serial.println("Grabbing thumbnail!");
      PTPThumbParser thumbParse;
      Serial.println("Picture transfer start");
      Serial.println(infoParse.picSize,DEC);
      PS.GetObject(prs.varBuffer[bufIndex], &thumbParse);
      //May need to send Lucas an end sequence.
      Serial.println("Done grabbing picture");
      //delay(5000);
    }
  }
  else if(incomingByte == 'S')
  {
    incomingByte = 0;
    Serial.println("Command received. Shutting down!");
    PS.Shutdown();
  }
  //Deletes the newest file on the camera
  else if(incomingByte == 'D')
```

```
  {
    incomingByte = 0;
    Serial.println("Command received. Deleting the newest object");
    PTPObjHandleParser prs;
    PS.GetObjectHandles(0xFFFFFFFF,0,0,&prs);

    int bufIndex = prs.varBuffer[16] - 1;
    Serial.println("Buffer index is:");
    Serial.println(bufIndex,DEC);

    Serial.println("Deleting the newest picture");
    PS.DeleteObject(prs.varBuffer[bufIndex]);
    delay(2000);
  }
  else if(incomingByte =='B')
  {
    incomingByte = 0;
    Serial.println("Command received. Sending battery level");
    uint8_t level;
    //0xD003 is the Canon-specific operation code for battery level
    PS.GetDevicePropValue(0xD003,level);
    Serial.println(level,DEC);
  }

  else
  {
    //Serial.println("No command received this time.");
    Serial.println(".");
    //digitalWrite(13,LOW);
  }

  //delay(5000);
  Usb.Task();

}
```

```
//File: canonpschdk.cpp
//Author: Aaron Olson
//Written for SD1112

//This file defines the interface class between the PTP library,
//the Canon Hack Development Kit firmware, and the Arduino

#include "canonpschdk.h"

void PSStateHandlers::OnSessionOpenedState(PTP *ptp)
{
    if (FAILED(((CanonPSCHDK*)ptp)->Initialize(true)) )
        PTPTRACE("Initialization error\r\n");

    ptp->SetState(PTP_STATE_DEVICE_INITIALIZED);
}


CanonPSCHDK::CanonPSCHDK(USB *pusb, PTPStateHandlers *s)
: PTP(pusb, s)
{
}

uint16_t CanonPSCHDK::EventCheck(PTPReadParser *parser)
{
    uint16_t    ptp_error   = PTP_RC_GeneralError;
    OperFlags   flags       = { 0, 0, 0, 1, 1, 0 };

    if ( (ptp_error = Transaction(PS_OC_CheckEvent, &flags, NULL, parser)) != PTP_RC_OK)
        PTPTRACE2("EOSEventCheck error: ", ptp_error);

    return ptp_error;
}

uint16_t CanonPSCHDK::Initialize(bool binit)
{
    uint16_t    ptp_error;

    if (binit)
    {
        if ((ptp_error = SetRecordMode(1)) != PTP_RC_OK)
                {
            PTPTRACE2("StartShootingMode failed: ", ptp_error);
                        return ptp_error;
                }
    }
    else
    {
        if ((ptp_error = SetRecordMode(0)) != PTP_RC_OK)
                {
            PTPTRACE2("EndShootingMode failed: ", ptp_error);
                        return ptp_error;
                }
```

```
    }

        //Sets zoom speed to 100%
        ptp_error = ExecLua("set_zoom_speed(100);");
    return ptp_error;
}

//Initiate a capture on the camera by invoking a CHDK-interpreted Lua script over PTP
uint16_t CanonPSCHDK::Capture()
{
    uint16_t    ptp_error;
    char script[] = "shoot();";
    OperFlags flags = { 2, 2, 1, 1, 3, 0};
    uint32_t params[2];

    params[0] = PTP_CHDK_ExecuteScript;
    params[1] = PTP_CHDK_SL_LUA;

    flags.dataSize = 9;//script.length()+1;

    ptp_error = Transaction(PTP_OC_CHDK, &flags, params, script);

    return ptp_error;
}

//Puts the camera into record mode
uint16_t CanonPSCHDK::SetRecordMode(int mode)
{
        uint16_t ptp_error;
        char script[] = "switch_mode_usb(1);";

        //Make sure a valid mode is passed.
        //Set the script to zero or one depending on passed param
        if(mode == 1)
        {
            script[16] = '1';
        }
        else if(mode == 0)
        {
            script[16] = '0';
        }
        else
        {
            ptp_error = 9999;
            PTPTRACE2("Invalid mode specified",ptp_error);
            return ptp_error;
        }

        OperFlags flags = { 2, 2, 1, 1, 3, 0};
        uint32_t params[2];

        params[0] = PTP_CHDK_ExecuteScript;
    params[1] = PTP_CHDK_SL_LUA;
```

```
    flags.dataSize = 20;//script.length()+1;

    ptp_error = Transaction(PTP_OC_CHDK, &flags, params, &script);

        return ptp_error;
}

//Sets the zoom level to something between 0 and 8 inclusive. The current camera only has 8
zoom steps
uint16_t CanonPSCHDK::SetZoom(int zoom)
{
        uint16_t ptp_error;
        OperFlags flags = { 2, 2, 1, 1, 3, 0};
        uint32_t params[2];

        params[0] = PTP_CHDK_ExecuteScript;
        params[1] = PTP_CHDK_SL_LUA;

        char script[] = "set_zoom(x);";
        flags.dataSize = 13; //length of the script

        if(zoom < 9)
        {
          char* tempchar;
          //Converts int in base 10 to char, stored in tempchar
          itoa(zoom,tempchar,10);
          script[9] = *(tempchar);
        }
        else
        {
          ptp_error = 9997;
          PTPTRACE2("Invalid zoom specified",ptp_error);
        }

        ptp_error = Transaction(PTP_OC_CHDK, &flags, params, script);

        return ptp_error;
}

uint16_t CanonPSCHDK::IncrementZoom(bool zoom)
{
        uint16_t ptp_error;
        OperFlags flags = {2, 2, 1, 1, 3, 0};
        uint32_t params[2];

        params[0] = PTP_CHDK_ExecuteScript;
        params[1] = PTP_CHDK_SL_LUA;

        char script[] = "set_zoom_rel(-1);";
        flags.dataSize = 18;

        if(zoom)
```

```
        {
          script[13] = '1';
          script[14] = ')';
          script[15] = ';';
          script[16] = '\0';
          flags.dataSize = 17;
        }

        ptp_error = Transaction(PTP_OC_CHDK, &flags, params, script);

        return ptp_error;
}

//Shuts down the camera
uint16_t CanonPSCHDK::Shutdown()
{
        uint16_t ptp_error;
        OperFlags flags = {2, 2, 1, 1, 3, 0};
        uint32_t params[2];

        params[0] = PTP_CHDK_ExecuteScript;
        params[1] = PTP_CHDK_SL_LUA;

        char script[] = "shut_down();";
        flags.dataSize = 13;

        ptp_error = Transaction(PTP_OC_CHDK, &flags, params, script);

}

//Runs a Lua script on the camera given by script
uint16_t CanonPSCHDK::ExecLua(String script)
{
        uint16_t ptp_error;
        OperFlags flags = { 2, 2, 1, 1, 3, 0};
        uint32_t params[2];

        params[0] = PTP_CHDK_ExecuteScript;
    params[1] = PTP_CHDK_SL_LUA;

    flags.dataSize = script.length()+1;
        if(flags.dataSize > 40)
        {
          ptp_error = 9998;
          PTPTRACE2("Script too long",ptp_error);
          return ptp_error;
        }

        char data[40];
        for(int i = 0;i<flags.dataSize;i++)
        {
          data[i] = script[i];
        }
```

```
    ptp_error = Transaction(PTP_OC_CHDK, &flags, params, data);

        return ptp_error;
}
```

```
/* Copyright (C) 2011 Circuits At Home, LTD. All rights reserved.

This software may be distributed and modified under the terms of the GNU
General Public License version 2 (GPL2) as published by the Free Software
Foundation and appearing in the file GPL2.TXT included in the packaging of
this file. Please note that GPL2 Section 2[b] requires that all works based
on this software must also be made publicly available under the terms of
the GPL2 ("Copyleft").

Contact information
-------------------

Circuits At Home, LTD
Web      :  http://www.circuitsathome.com
e-mail   :  support@circuitsathome.com
*/
#ifndef __CANONPSCHDK_H__
#define __CANONPSCHDK_H__

#include <ptp.h>
#include "chdkptp.h"

// PTP Operation Codes (PowerShot specific)
#define PS_OC_GetObjectSize             0x9001
#define PS_OC_StartShootingMode             0x9008
#define PS_OC_EndShootingMode               0x9009
#define PS_OC_ViewfinderOn              0x900B
#define PS_OC_ViewfinderOff             0x900C
#define PS_OC_ReflectChanges                0x900D
#define PS_OC_CheckEvent                0x9013
#define PS_OC_FocusLock                 0x9014
#define PS_OC_FocusUnlock               0x9015
#define PS_OC_InitiateCaptureInMemory               0x901A
#define PS_OC_GetPartialObject              0x901B
#define PS_OC_GetViewfinderImage            0x901d
#define PS_OC_GetChanges                0x9020
#define PS_OC_GetFolderEntries              0x9021

// PTP PowerShot Extention Events
#define PS_EC_ShutDownCFDoorWasOpened               0xC001      /* The Device has shut down due
to the opening of the SD card cover.*/
#define PS_EC_ResetHwError              0xC005      /* The device has generated a hardware
error. */
#define PS_EC_AbortPCEvf                0xC006      /* The Viewfinder mode has been cancelled. */
#define PS_EC_EnablePCEvf               0xC007      /* The Viewfinder mode has been enablede. */
#define PS_EC_FullViewReleased              0xC008      /* Transfer timing of main image data */
#define PS_EC_ThumbnailReleased             0xC009      /* Transfer timing of thumbnail image
data */
#define PS_EC_ChangeBatteryStatus           0xC00A      /* The power condition of the camera
has changed. */
#define PS_EC_PushedReleaseSw               0xC00B      /* User has pressed the release swtich
on camera. */
#define PS_EC_PropertyChanged               0xC00C      /* A group of properties relating to
```

```
release control have been changed. */
#define PS_EC_RotationAngleChanged          0xC00D      /* The angle of rotation of the camera
has been changed. */
#define PS_EC_ChangedByCamUI                0xC00E      /* An operation control on the camera
has been operated.*/
#define PS_EC_Shutdown                  0xD001      /* Shutdown */
#define PS_EC_StartDirectTransfer           0xC011
#define PS_EC_StopDirectTransfer            0xC013

// PowerShot-specific Device Properties
#define PS_DPC_BeepMode                 0xD001
#define PS_DPC_BatteryKind              0xD002
#define PS_DPC_BatteryStatus                0xD003
#define PS_DPC_UILockType               0xD004
#define PS_DPC_CameraMode               0xD005
#define PS_DPC_ImageQuality             0xD006
#define PS_DPC_FullViewFileFormat           0xD007
#define PS_DPC_ImageSize                0xD008
#define PS_DPC_SelfTime                 0xD009
#define PS_DPC_FlashMode                0xD00A
#define PS_DPC_Beep                 0xD00B
#define PS_DPC_ShootingMode             0xD00C
#define PS_DPC_ImageMode                0xD00D
#define PS_DPC_DriveMode                0xD00E
#define PS_DPC_EZoom                    0xD00F
#define PS_DPC_MeteringMode             0xD010
#define PS_DPC_AFDistance               0xD011
#define PS_DPC_FocusingPoint                0xD012
#define PS_DPC_WhiteBalance             0xD013
#define PS_DPC_SlowShutterSetting           0xD014
#define PS_DPC_AFMode                   0xD015
#define PS_DPC_ImageStabilization           0xD016
#define PS_DPC_Contrast                 0xD017
#define PS_DPC_ColorGain                0xD018
#define PS_DPC_Sharpness                0xD019
#define PS_DPC_Sensitivity              0xD01A
#define PS_DPC_ParameterSet             0xD01B
#define PS_DPC_ISOSpeed                 0xD01C
#define PS_DPC_Aperture                 0xD01D
#define PS_DPC_ShutterSpeed             0xD01E
#define PS_DPC_ExpCompensation              0xD01F
#define PS_DPC_FlashCompensation            0xD020
#define PS_DPC_AEBExposureCompensation              0xD021
#define PS_DPC_AvOpen                   0xD023
#define PS_DPC_AvMax                    0xD024
#define PS_DPC_FocalLength              0xD025
#define PS_DPC_FocalLengthTele              0xD026
#define PS_DPC_FocalLengthWide              0xD027
#define PS_DPC_FocalLengthDenominator               0xD028
#define PS_DPC_CaptureTransferMode          0xD029
#define PS_DPC_Zoom                 0xD02A
#define PS_DPC_NamePrefix               0xD02B
#define PS_DPC_SizeQualityMode              0xD02C
```

```
#define PS_DPC_SupportedThumbSize           0xD02D
#define PS_DPC_SizeOfOutputDataFromCamera           0xD02E
#define PS_DPC_SizeOfInputDataToCamera              0xD02F
#define PS_DPC_RemoteAPIVersion             0xD030
#define PS_DPC_FirmwareVersion              0xD031
#define PS_DPC_CameraModel              0xD032
#define PS_DPC_CameraOwner              0xD033
#define PS_DPC_UnixTime                 0xD034
#define PS_DPC_CameraBodyID             0xD035
#define PS_DPC_CameraOutput             0xD036
#define PS_DPC_DispAv                   0xD037
#define PS_DPC_AvOpenApex               0xD038
#define PS_DPC_DZoomMagnification           0xD039
#define PS_DPC_MlSpotPos                0xD03A
#define PS_DPC_DispAvMax                0xD03B
#define PS_DPC_AvMaxApex                0xD03C
#define PS_DPC_EZoomStartPosition           0xD03D
#define PS_DPC_FocalLengthOfTele            0xD03E
#define PS_DPC_EZoomSizeOfTele              0xD03F
#define PS_DPC_PhotoEffect              0xD040
#define PS_DPC_AssistLight              0xD041
#define PS_DPC_FlashQuantityCount           0xD042
#define PS_DPC_RotationAngle                0xD043
#define PS_DPC_RotationScene                0xD044
#define PS_DPC_EventEmulateMode             0xD045
#define PS_DPC_DPOFVersion              0xD046
#define PS_DPC_TypeOfSupportedSlideShow             0xD047
#define PS_DPC_AverageFilesizes             0xD048
#define PS_DPC_ModelID                  0xD049


class PSStateHandlers : public PTPStateHandlers
{
public:
      virtual void OnSessionOpenedState(PTP *ptp);
};



class CanonPSCHDK : public PTP
{
public:
    // ISO Speed Values
    enum { IsoAuto = 0, Iso80 = 0x45, Iso100 = 0x48, Iso200 = 0x50, Iso400 = 0x58, Iso800=0x60 };

    // White Balance Values
    enum { WbAuto = 0, WbSunny, WbCloudy, WbTungsten, WbFluorescent, WbFlash, WbCustom,
    WbUnknown };

    // Exposure Compensation Values (same values for both exposure compensation and flash
    compensation)
    enum {  ExpCompDown2    = 0x08, ExpCompDown1_2d3= 0x0B, ExpCompDown1_1d3= 0x0D, ExpCompDown1
     = 0x10, ExpCompDown2d3 = 0x13,
```

```
            ExpCompDown1d3  = 0x15, ExpComp_0       = 18,   ExpCompUp1d3    = 0x1B, ExpCompUp2d3
             = 0x1D, ExpCompUp1     = 0x20,
            ExpCompUp1_1d3  = 0x23, ExpCompUp1_2d3  = 0x25, ExpCompUp2      = 0x28 };

    // Image Quality Values
    enum { ImgQualityNormal = 2, ImageQualityFine = 3, ImageQualitySuperb = 5 };

    // Image Size Values
    enum { ImgSizeLarge, ImgSizeMedium1, ImgSizeSmall, ImgSizeMedium2 };

    CanonPSCHDK(USB *pusb, PTPStateHandlers *s);

    uint16_t Initialize(bool binit);

    uint16_t Capture();
        uint16_t SetRecordMode(int mode);
        uint16_t ExecLua(String script);
        uint16_t SetZoom(int zoom);
        uint16_t IncrementZoom(bool zoom);
        uint16_t Shutdown();
    uint16_t EventCheck(PTPReadParser *parser);
};

#endif // __CANONPS_H__
```

```
#ifndef __CHDK_PTP_H
#define __CHDK_PTP_H
#define PTP_CHDK_VERSION_MAJOR 2  // increase only with backwards incompatible changes (and
reset minor)
#define PTP_CHDK_VERSION_MINOR 1  // increase with extensions of functionality
/*
protocol version history
0.1 - initial proposal from mweerden, + luar
0.2 - Added ScriptStatus and ScriptSupport, based on work by ultimA
1.0 - removed old script result code (luar), replace with message system
2.0 - return PTP_CHDK_TYPE_TABLE for tables instead of TYPE_STRING, allow return of empty strings
*/

#define PTP_OC_CHDK 0x9999

// PTP constants - only need to be defined in CHDK, not PC side
#ifndef PTP_RC_OK
#define PTP_RC_OK 0x2001
#define PTP_RC_GeneralError 0x2002
#define PTP_RC_ParameterNotSupported 0x2006
#define PTP_RC_InvalidParameter 0x201D
#endif

// N.B.: unused parameters should be set to 0
enum ptp_chdk_command {
  PTP_CHDK_Version = 0,     // return param1 is major version number
                            // return param2 is minor version number
  PTP_CHDK_GetMemory,       // param2 is base address (not NULL; circumvent by taking
  0xFFFFFFFF and size+1)
                            // param3 is size (in bytes)
                            // return data is memory block
  PTP_CHDK_SetMemory,       // param2 is address
                            // param3 is size (in bytes)
                            // data is new memory block
  PTP_CHDK_CallFunction,    // param2 are flags: 0x1 means use rest of params for pointer and
  args to allow function to send back data
                            // (return) data is either:
                            //   - array of function pointer and (long) arguments if flag 0x1
                            is not set  (max: 10 args)
                            //   - return data provided by called function if flag 0x1 is set
                            // return param1 is return value
  PTP_CHDK_TempData,        // data is data to be stored for later
                            // param2 is for the TD flags below
  PTP_CHDK_UploadFile,      // data is 4-byte length of filename, followed by filename and
  contents
  PTP_CHDK_DownloadFile,    // preceded by PTP_CHDK_TempData with filename
                            // return data are file contents
  PTP_CHDK_ExecuteScript,   // data is script to be executed
                            // param2 is language of script
                            // return param1 is script id, like a process id
                            // return param2 is status, PTP_CHDK_S_ERRTYPE*
  PTP_CHDK_ScriptStatus,    // Script execution status
                            // return param1 bits
```

```
                            // PTP_CHDK_SCRIPT_STATUS_RUN is set if a script running, cleared
                            if not
                            // PTP_CHDK_SCRIPT_STATUS_MSG is set if script messages from script
                            waiting to be read
                            // all other bits and params are reserved for future use
  PTP_CHDK_ScriptSupport,   // Which scripting interfaces are supported in this build
                            // param1 CHDK_PTP_SUPPORT_LUA is set if lua is supported, cleared
                            if not
                            // all other bits and params are reserved for future use
  PTP_CHDK_ReadScriptMsg,   // read next message from camera script system
                            // return param1 is chdk_ptp_s_msg_type
                            // return param2 is message subtype:
                            //   for script return and users this is ptp_chdk_script_data_type
                            //   for error ptp_chdk_script_error_type
                            // return param3 is script id of script that generated the message
                            // return param4 is length of the message data.
                            // return data is message.
                            // A minimum of 1 bytes of zeros is returned if the message has no
                            data (empty string or type NONE)
  PTP_CHDK_WriteScriptMsg,  // write a message for scripts running on camera
                            // input param2 is target script id, 0=don't care. Messages for a
                            non-running script will be discarded
                            // data length is handled by ptp data phase
                            // input messages do not have type or subtype, they are always a
                            string destined for the script (similar to USER/string)
                            // output param1 is ptp_chdk_script_msg_status
};

// data types as used by ReadScriptMessage
enum ptp_chdk_script_data_type {
  PTP_CHDK_TYPE_UNSUPPORTED = 0, // type name will be returned in data
  PTP_CHDK_TYPE_NIL,
  PTP_CHDK_TYPE_BOOLEAN,
  PTP_CHDK_TYPE_INTEGER,
  PTP_CHDK_TYPE_STRING, // Empty strings are returned with length=0
  PTP_CHDK_TYPE_TABLE,  // tables are converted to a string by usb_msg_table_to_string,
                        // this function can be overridden in lua to change the format
                        // the string may be empty for an empty table
};

// TempData flags
#define PTP_CHDK_TD_DOWNLOAD  0x1  // download data instead of upload
#define PTP_CHDK_TD_CLEAR     0x2  // clear the stored data; with DOWNLOAD this
                                   // means first download, then clear and
                                   // without DOWNLOAD this means no uploading,
                                   // just clear

// Script Languages - for execution only lua is supported for now
#define PTP_CHDK_SL_LUA    0
#define PTP_CHDK_SL_UBASIC 1

// bit flags for script status
#define PTP_CHDK_SCRIPT_STATUS_RUN   0x1 // script running
```

```
#define PTP_CHDK_SCRIPT_STATUS_MSG   0x2 // messages waiting
// bit flags for scripting support
#define PTP_CHDK_SCRIPT_SUPPORT_LUA  0x1

// message types
enum ptp_chdk_script_msg_type {
    PTP_CHDK_S_MSGTYPE_NONE = 0, // no messages waiting
    PTP_CHDK_S_MSGTYPE_ERR,      // error message
    PTP_CHDK_S_MSGTYPE_RET,      // script return value
    PTP_CHDK_S_MSGTYPE_USER,     // message queued by script
// TODO chdk console data ?
};

// error subtypes for PTP_CHDK_S_MSGTYPE_ERR and script startup status
enum ptp_chdk_script_error_type {
    PTP_CHDK_S_ERRTYPE_NONE = 0,
    PTP_CHDK_S_ERRTYPE_COMPILE,
    PTP_CHDK_S_ERRTYPE_RUN,
};

// message status
enum ptp_chdk_script_msg_status {
    PTP_CHDK_S_MSGSTATUS_OK = 0, // queued ok
    PTP_CHDK_S_MSGSTATUS_NOTRUN, // no script is running
    PTP_CHDK_S_MSGSTATUS_QFULL,  // queue is full
    PTP_CHDK_S_MSGSTATUS_BADID,  // specified ID is not running
};
#endif // __CHDK_PTP_H
```

```
#include "pseventparser.h"

void PSEventParser::Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset)
{
    uint16_t    cntdn   = (uint16_t)len;
    uint8_t     *p      = (uint8_t*)pbuf;

    switch (nStage)
    {
    case 0:
        p       += 12;
        cntdn   -= 12;

        if (!cntdn)
            return;
        nStage  ++;

    case 1:
        //Notify(PSTR("\r\nEvent Block Size:\t"));
        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage ++;
    case 2:
        if (!valueParser.Parse(&p, &cntdn))
            return;

        //PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
        nStage ++;
    case 3:
        //Notify(PSTR("\r\nNumber of Fields:\t"));
        theBuffer.valueSize = 2;
        valueParser.Initialize(&theBuffer);
        nStage ++;
    case 4:
        if (!valueParser.Parse(&p, &cntdn))
            return;

        //PrintHex<uint16_t>(*((uint16_t*)theBuffer.pValue));
        nStage ++;
    case 5:
        //Notify(PSTR("\r\nEvent Code:\t"));
        theBuffer.valueSize = 2;
        valueParser.Initialize(&theBuffer);
        nStage ++;
    case 6:
        if (!valueParser.Parse(&p, &cntdn))
            return;

        eventCode = *((uint16_t*)theBuffer.pValue);
        //PrintHex<uint16_t>(*((uint16_t*)theBuffer.pValue));
        nStage ++;
    case 7:
        //Notify(PSTR("\r\nTransaction ID:\t"));
```

```
        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage ++;
    case 8:
        if (!valueParser.Parse(&p, &cntdn))
            return;

        //PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
        nStage ++;
    case 9:
        if (eventCode == PTP_EC_ObjectAdded)
            Notify(PSTR("\r\nObject Added:\t\t"));

        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage ++;
    case 10:
        if (eventCode == PTP_EC_ObjectAdded)
        {
            if (!valueParser.Parse(&p, &cntdn))
                return;

            objHandle = *((uint32_t*)theBuffer.pValue);
            PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
            Notify(PSTR("\r\n"));
        }
        if (eventCode == PTP_EC_CaptureComplete)
            Notify(PSTR("\r\nCapture complete.\r\n"));
        nStage ++;
    case 11:
        nStage = 0;
    }
}
```

```
#ifndef __PSEVENTPARSER_H__
#define __PSEVENTPARSER_H__

#include <inttypes.h>
#include <avr/pgmspace.h>
#include <message.h>
#include <parsetools.h>
#include "ptpdebug.h"
#include "canonpschdk.h"

class PSEventParser : public PTPReadParser
{
    MultiValueBuffer                    theBuffer;
    uint32_t                            varBuffer;
    uint8_t                             nStage;
        uint16_t                            eventCode;
    uint32_t                            objHandle;

    MultiByteValueParser                valueParser;

public:
    PSEventParser() : nStage(0), varBuffer(0), objHandle(0)
    {
        theBuffer.pValue = &varBuffer;
    };
    uint32_t GetObjHandle() { return objHandle; };
    virtual void Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset);
};

#endif // __PSEVENTPARSER_H__
```

```
//File: ptpobjhandleparser.cpp
//Author: Aaron Olson
//Written for SD1112

//This class parses the object handles when they are requested from the camera. The
//highest-numbered object handle belongs to the newest file.

#include "ptpobjhandleparser.h"

void PTPObjHandleParser::Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset)
{
    uint16_t cntdn = (uint16_t)len;
    uint32_t *p = (uint32_t*)pbuf;

        //Save a copy of the buffer from last time through
        for(int i = 0; i<17; i++)
        {
          prevBuf[i] = inBuf[i];
        }


      for(int i = 0; i<16; i++)
      {
        inBuf[i] = p[i];
      }

      //Finds the index of the last element in the buffer.
      inBuf[16] = (uint32_t)cntdn/4;

      varBuffer = inBuf;

      if(inBuf[16] == 0)
      {
        if(inBuf[0] == 0)
        {
          prevBuf[16] = 16;
          varBuffer = prevBuf;
        }
        else
          varBuffer = inBuf;
      }
      else
      {
        varBuffer = inBuf;
      }
}
```

```
#ifndef __PTPOBJHANDLEPARSER_H__
#define __PTPOBJHANDLEPARSER_H__

#include <inttypes.h>
#include <avr/pgmspace.h>
//#include <printhex.h>
//#include <message.h>
//#include <hexdump.h>
#include <parsetools.h>
#include "ptpcallback.h"
#include "ptpdebug.h"

class PTPObjHandleParser : public PTPReadParser
{
    MultiValueBuffer theBuffer;

    uint8_t nStage;
    PTPListParser arrayParser;


public:
    //PTPObjHandleParser() : nStage(0) { arrayParser.Initialize(4, 4, &theBuffer); };
    virtual void Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset);

        uint32_t inBuf[17];
        uint32_t prevBuf[17];
        uint32_t* varBuffer;
};


#endif // __PTPOBJHANDLEPARSER_H__
```

```
//File: ptpobjinfoparser.cpp
//Adapted from the Arduino PTP library by Aaron Olson for SD1112

#include "ptpobjinfoparser.h"

const char* PTPObjInfoParser::acNames[] PROGMEM =
{
    msgUndefined,
    msgAssociation,
    msgScript,
    msgExecutable,
    msgText,
    msgHTML,
    msgDPOF,
    msgAIFF,
    msgWAV,
    msgMP3,
    msgAVI,
    msgMPEG,
    msgASF,
    msgQT
};

const char* PTPObjInfoParser::imNames[] PROGMEM =
{
    msgUndefined,
    msgEXIF_JPEG,
    msgTIFF_EP,
    msgFlashPix,
    msgBMP,
    msgCIFF,
    msgUndefined_0x3806,
    msgGIF,
    msgJFIF,
    msgPCD,
    msgPICT,
    msgPNG,
    msgUndefined_0x380C,
    msgTIFF,
    msgTIFF_IT,
    msgJP2,
    msgJPX,
};

void PTPObjInfoParser::PrintFormat(uint16_t op)
{
    Serial.print(op, HEX);
    Serial.print("\t");
    //Notify(msgTab);

    if ((((op >> 8) & 0xFF) == 0x30) && ((op & 0xFF) <= (PTP_OFC_QT & 0xFF)))
        Notify((char*)pgm_read_word(&acNames[(op & 0xFF)]));
    else
```

```
        if ((((op >> 8) & 0xFF) == 0x38) && ((op & 0xFF) <= (PTP_OFC_JPX & 0xFF)))
            Notify((char*)pgm_read_word(&imNames[(op & 0xFF)]));
        else
        {
            switch (op)
            {
            case MTP_OFC_Undefined_Firmware:
                Notify(msgUndefined_Firmware);
                break;
            case MTP_OFC_Windows_Image_Format:
                Notify(msgWindows_Image_Format);
                break;
            case MTP_OFC_Undefined_Audio:
                Notify(msgUndefined_Audio);
                break;
            case MTP_OFC_WMA:
                Notify(msgWMA);
                break;
            case MTP_OFC_OGG:
                Notify(msgOGG);
                break;
            case MTP_OFC_AAC:
                Notify(msgAAC);
                break;
            case MTP_OFC_Audible:
                Notify(msgAudible);
                break;
            case MTP_OFC_FLAC:
                Notify(msgFLAC);
                break;
            case MTP_OFC_Undefined_Video:
                Notify(msgUndefined_Video);
                break;
            case MTP_OFC_WMV:
                Notify(msgWMV);
                break;
            case MTP_OFC_MP4_Container:
                Notify(msgMP4_Container);
                break;
            case MTP_OFC_MP2:
                Notify(msgMP2);
                break;
            case MTP_OFC_3GP_Container:
                Notify(msg3GP_Container);
                break;
            default:
                Notify(PSTR("Vendor defined"));
            }
        }
    Notify(PSTR("\r\n"));
}

void PTPObjInfoParser::Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset)
```

```
{
    uint16_t    cntdn   = (uint16_t)len;
    uint8_t     *p  = (uint8_t*)pbuf;

    switch (nStage)
    {
    case 0:
        p   += 12;
        cntdn   -= 12;
        nStage  ++;
    case 1:
        //Notify(PSTR("Storage ID:\t\t"));
        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 2:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
        nStage  ++;
    case 3:
        //Notify(PSTR("\r\nObject Format:\t\t"));
        theBuffer.valueSize = 2;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 4:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintFormat(*((uint16_t*)theBuffer.pValue));
                pic = *((uint16_t*)theBuffer.pValue);
        nStage  ++;
    case 5:
        //Notify(PSTR("Protection Status:\t"));
        theBuffer.valueSize = 2;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 6:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintHex<uint16_t>(*((uint16_t*)theBuffer.pValue));
        nStage  ++;
    case 7:
        //Notify(PSTR("\r\nObject Compressed Size:\t"));
        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 8:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
                picSize = *((uint32_t*)theBuffer.pValue);
        nStage  ++;
    case 9:
```

```
        //Notify(PSTR("\r\nThumb Format:\t\t"));
        theBuffer.valueSize = 2;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 10:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintFormat(*((uint16_t*)theBuffer.pValue));
                thumb = *((uint16_t*)theBuffer.pValue);
        nStage++;
    case 11:
        //Notify(PSTR("Thumb Compressed Size:\t"));
        theBuffer.valueSize = 4;
        valueParser.Initialize(&theBuffer);
        nStage  ++;
    case 12:
        if (!valueParser.Parse(&p, &cntdn))
            return;
        //PrintHex<uint32_t>(*((uint32_t*)theBuffer.pValue));
                thumbSize = *((uint16_t*)theBuffer.pValue);
        nStage  ++;

    }
}
```

```
#ifndef __PTPOBJINFOPARSER_H__
#define __PTPOBJINFOPARSER_H__

#include <message.h>
#include <parsetools.h>

#include <ptp.h>
#include <mtpconst.h>

const char msgUndefined     [] PROGMEM = "Undefined";

// Ancillary formats
const char msgAssociation           [] PROGMEM = "Association";
const char msgScript                [] PROGMEM = "Script";
const char msgExecutable            [] PROGMEM = "Executable";
const char msgText          [] PROGMEM = "Text";
const char msgHTML          [] PROGMEM = "HTML";
const char msgDPOF          [] PROGMEM = "DPOF";
const char msgAIFF          [] PROGMEM = "AIFF";
const char msgWAV           [] PROGMEM = "WAV";
const char msgMP3           [] PROGMEM = "MP3";
const char msgAVI           [] PROGMEM = "AVI";
const char msgMPEG          [] PROGMEM = "MPEG";
const char msgASF           [] PROGMEM = "ASF";
const char msgQT            [] PROGMEM = "QT";

// Image formats
const char msgEXIF_JPEG         [] PROGMEM = "EXIF_JPEG";
const char msgTIFF_EP           [] PROGMEM = "TIFF_EP";
const char msgFlashPix          [] PROGMEM = "FlashPix";
const char msgBMP           [] PROGMEM = "BMP";
const char msgCIFF          [] PROGMEM = "CIFF";
const char msgUndefined_0x3806          [] PROGMEM = "Undefined_0x3806";
const char msgGIF           [] PROGMEM = "GIF";
const char msgJFIF          [] PROGMEM = "JFIF";
const char msgPCD           [] PROGMEM = "PCD";
const char msgPICT          [] PROGMEM = "PICT";
const char msgPNG           [] PROGMEM = "PNG";
const char msgUndefined_0x380C          [] PROGMEM = "Undefined_0x380C";
const char msgTIFF          [] PROGMEM = "TIFF";
const char msgTIFF_IT           [] PROGMEM = "TIFF_IT";
const char msgJP2           [] PROGMEM = "JP2";
const char msgJPX           [] PROGMEM = "JPX";

// MTP Object Formats
const char msgUndefined_Firmware    [] PROGMEM = "Undefined_Firmware";
const char msgWindows_Image_Format  [] PROGMEM = "Windows_Image_Format";
const char msgUndefined_Audio       [] PROGMEM = "Undefined_Audio";
const char msgWMA           [] PROGMEM = "WMA";
const char msgOGG           [] PROGMEM = "OGG";
const char msgAAC           [] PROGMEM = "AAC";
const char msgAudible           [] PROGMEM = "Audible";
const char msgFLAC          [] PROGMEM = "FLAC";
```

```
const char msgUndefined_Video       [] PROGMEM = "Undefined_Video";
const char msgWMV           [] PROGMEM = "WMV";
const char msgMP4_Container     [] PROGMEM = "MP4_Container";
const char msgMP2           [] PROGMEM = "MP2";
const char msg3GP_Container     [] PROGMEM = "3GP_Container";


class PTPObjInfoParser : public PTPReadParser
{
    static const char* acNames[];
    static const char* imNames[];

    MultiValueBuffer theBuffer;
    uint32_t varBuffer;
    uint8_t nStage;

    MultiByteValueParser valueParser;
    PTPListParser arrayParser;

    static void PrintChar(MultiValueBuffer *p)
    {
        if (((unsigned char*)p->pValue)[0])
            Serial.print(((unsigned char*)p->pValue)[0]);
    };
    void PrintFormat(uint16_t op);

public:
    PTPObjInfoParser() : nStage(0) { theBuffer.pValue = (uint8_t*)&varBuffer; };
    virtual void Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset);

        int thumb;
        int pic;
        int thumbSize;
        uint32_t picSize;
};

#endif // __PTPOBJINFOPARSER_H__
```

```
//File: ptpthumbparser.cpp
//Author: Aaron Olson
//Written for SD1112

#include "ptpthumbparser.h"

void PTPThumbParser::Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset)
{
    uint8_t cntdn = (uint16_t)len;
    uint8_t *p = (uint8_t*)pbuf;

    for(int i = 0; i<cntdn; i++)
    {
        Serial.write(p[i]);
    }

}
```

```
#ifndef __PTPTHUMBPARSER_H__
#define __PTPTHUMBPARSER_H__

#include <inttypes.h>
#include <avr/pgmspace.h>
#include <parsetools.h>
#include "ptpcallback.h"
#include "ptpdebug.h"

class PTPThumbParser : public PTPReadParser
{
    MultiValueBuffer theBuffer;
    uint8_t nStage;
    PTPListParser arrayParser;

    public:
        virtual void Parse(const uint16_t len, const uint8_t *pbuf, const uint32_t &offset);

};


#endif
```